SAND2005-3413C
Unlimited Release
May 2005

Characterizing Compiler Performance for the AMD Opteron Processor on a Parallel Platform

Douglas Doerfler
Courtenay Vaughan
Sandia National Laboratories

Abstract

Application performance on a high performance, parallel platform depends on a variety of factors, the most important being the performance of the high speed interconnect and the compute node processor. The performance of the compute processor depends on how well the compiler optimizes for a given processor architecture, and how well it optimizes the applications source code. An analysis of uni-processor and parallel performance using different AMD Opteron compilers on key SNL application codes is presented.

1 Introduction

The Advanced Micro Devices (AMD) Opteron Processor has been the processor of choice for a wide variety of new scientific computing platforms, including Sandia National Laboratories (SNL) Red Storm, and the Cray XT3 and XD1 architectures. The nominal compiler for Red Storm is the Portland Group (PGI) C, C++ and Fortran compiler suite. SNL has also investigated the Pathscale EKOPath compiler suite. This paper looks at the performance of these two compilers on application codes of interest at SNL: CTH, LAMMPS and PARTISN.

There are other excellent C, C++ and Fortran compilers on the market not included in this study. The results of this study are due to an SNL evaluation of the Pathscale product and its potential use in present and future projects. Other compilers may be evaluated and if so those results may be published at a later date. Although SNL has a very close relationship with both PGI and Pathscale, for this study neither company was consulted in order to help optimize for a particular application. We wanted to present results that would be obtained by a typical application developer with experience in optimization methods and techniques, but without the help and expertise of the vendor.

Most compiler evaluations look at single processor performance. We will also look at scaled (weak-scaling) problem sets for each application and analyze the performance differences as a function of scale. By analyzing the differences in scaled run times between two versions of the same application but with different levels of optimization, it is possible to infer about the relationship of computation and communication for a given application.

2 The Effect of Improving Computational Efficiency

Total execution time at a given scale can be significantly affected by factors other than the processors efficiency at performing scientific calculations. Normally there is overhead associated with the parallel implementation of a given algorithm, which includes the time spent moving the data between nodes. As the problem scales, the overhead may increase and the parallel efficiency decreases. Therefore, as scale increases a decrease in time spent computing due to better compiler optimizations may not provide as large a decrease in overall run time. The total fraction of time spent in a given application at scale N is defined by

$$F(N) = f_c(N) + f_o(N) = 1$$

Where f_c is the fraction of time spent in computation, f_o is the fraction of time devoted to overhead (parallel inefficiencies) and N is the number of processors involved in the calculation. An increase in computational efficiency will only effect the f_c term.

$$F_s(N) = f_c(N) * f_s(N) + f_o(N) < 1$$

Where F_s is defined to be the fractional speedup of the total run time and f_s is defined to be the fractional speedup of the computational part of the total time. The application measures and reports the run time for a given metric, such as an iteration time or a grind time. The term F_s is calculated by taking the ratio of the times reported by each version.

$$F_s(N) = \text{(time for faster runtime)} / \text{(time for slower runtime)}$$

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

For this study, percent speedup is reported and it is defined to be

$$\text{Speedup (\%)} = (1 / F_s(N) - 1) * 100$$

For each application the speedup is plotted as a function of N. If an application provides perfect scaling, for example no extra work is involved as the problem is partitioned to fit on N nodes and the communication completely overlaps with the computation, then the f_o term would be zero and all compiler optimizations would be realized at all scale levels. If an application has the characteristic of f_c and f_o being constant as N increases, then F_s will be independent of scale and the application will take advantage of optimizations made by a given compiler, although F_s will be less than f_s. If f_o increases as a function of scale, but f_c remains constant then F_s will increase as the size of the job grows and the effect of the compiler optimizations will decrease.

3 Compiler Suites

Both compilers have switch settings that provide a higher level of abstraction for commonly used, safe optimizations. Further optimization can be had by exploiting knowledge about the applications operation and choosing the appropriate optimization flags. In addition, optimizations can be made that may reduce the accuracy of the results and inter-procedural analysis optimizations can also be performed. For this study, inter-procedural analysis optimizations were not enabled. Primarily due to the long compile times associated with enabling this level of optimization.

3.1 The PGI Compiler Suite

PGI was selected early on by AMD as a partner for its Opteron Processor line. In addition, SNL has been using PGI's optimizing compilers on its ASCI Red platform for several years and all of Sandia's key applications have been ported to the PGI suite. As such, it was logical to select PGI as the nominal compiler for Red Storm. The PGI compiler suite has a wide variety of optimization flags that can be used to tune performance to a particular application. However, we have found that a good starting point, and in most cases a good ending point also, is to use the "-fastsse" switch. This switch setting provides a higher level of abstraction for the following flags.

```
-fastsse ->    -fast -Mvect=sse -Mscalarsse -Mcache_align –Mflushz
-fast ->       -O2 -Munroll=c:1 -Mnoframe –Mlre
```

It includes the "-fast" switch which sets optimization to level 2, unrolls loops, does not use a true stack frame pointer for functions, and enables loop-carried redundancy elimination. In addition, "-fastsse" enables the SSE, SSE2 and 3Dnow instructions, including special prefetch instructions, aligns unconstrained data objects on cache line boundaries, and sets SSE to "flush-to-zero" mode.

All results presented are with optimization set to "-fastsse" unless otherwise noted. PGI version 6.0 was used for this study. [1]

3.2. Pathscale Compiler Suite

Pathscale is a relatively new player in the compiler market, but their compiler shares a heritage with the Silicon Graphics compiler suite. Pathscale was very interested in working with SNL early on in their development process to help debug and validate their compilers. The work presented is independent of that effort, but perhaps some of the optimizations they have chosen to make are due to the collaboration.

As with the PGI compiler, we have found a good starting point for the Pathscale compiler flag with the options, "-O3 -OPT:Ofast". In addition to optimization level 3, the "-OPT:Ofast" option provides:

```
-OPT:Ofast ->    -OPT:ro=2:Olimit=0:div_split=ON:alias=typed -msse2
```

The round off level is set to 2, which allows extensive transformations to be made which may cause some differences in floating point results as compared to a strict IEEE conformance implementation. The "Olimit" option tells the compiler to optimize no matter what the file size. The "div_split" option changes a division operation to x*(reciprocal of y). In addition, SSE2 instructions are enabled. The round off option, although declared explicitly with Pathscale, is implied by the PGI compiler due to the lack of the "-Kieee" option. Thus, each compiler is making run time performance optimizations at the cost of floating point precision.

All results presented are with optimization set to “-O3 –OPT:Ofast” unless otherwise noted. Version 2.1 of the compiler suite was used for this study. [2]

4 The Test Platform

All tests were conducted on SNL’s Red Squall cluster. Specifications of Red Squall are given in Table 1. Although each node is a dual-Opteron motherboard, only one application process per node was allowed for this study in order to eliminate effects associated with mixing intra-node and inter-node message passing issues. Tests were scaled up to 128 nodes for each application.

Table 1: Red Squall Specifications

Compute Node	Dual Processor 2.0 GHz Opteron with 2 GBytes of 333Mhz DDR DRAM per processor
Operating System	SuSE Linux Professional 9.0
High Speed Interconnect	Quadrics Elan4 w/PCI-X, single rail
MPI	MPICH 1.2.4 w/Quadrics extensions
PGI Compiler	Version 6.0
Pathscale Compiler	Version 2.1

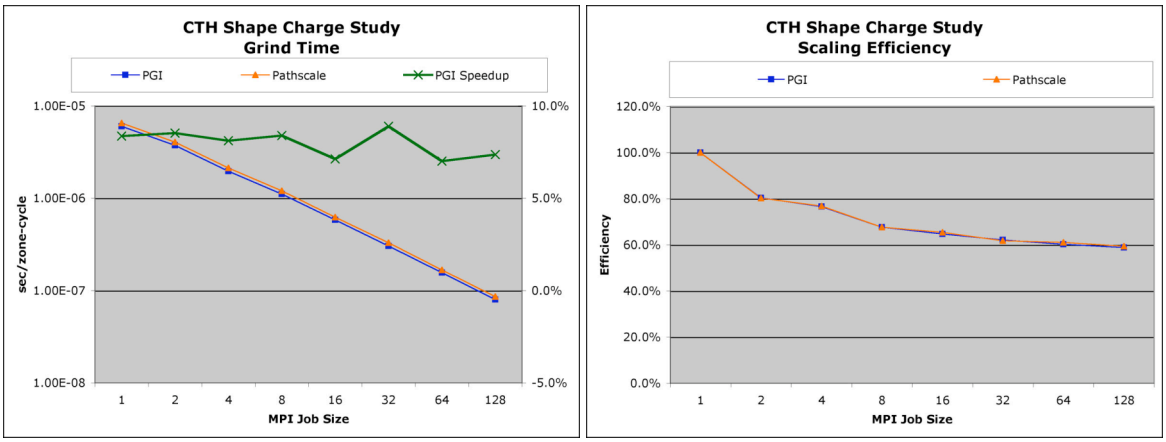
5 Applications

All application results are for a scaled problem set. That is, each processor has approximately the same amount of data to process independent of the job size. For an ideal environment, the time of execution would be the same for all job sizes. Of course, many real world effects limit the scalability of a given platform. So as the size of a job increases, in general the run time also increases. This is primarily due to the ability of the platform to effectively move data between the nodes. In addition, there are application and problem set effects. Most SNL applications operate on three-dimensional volumes, and a subset of the problem size is allocated to each node in the job. As a problem set is scaled to a larger set of nodes, the volume allocation may not be cubic. That is one dimension of the volume may grow in size, but the other two dimensions remain fixed. The decision on how the data scaling takes place effects how the data is partitioned across all the nodes in the job and hence affects the surface to volume calculations.

For each test, the average time of three runs is reported.

5.1 CTH

CTH is a multi-material, large deformation, strong shock wave, solid mechanics code. CTH has models for multi-phase, elastic viscoplastic, porous and explosive materials. Three-dimensional rectangular meshes, two-dimensional rectangular and cylindrical meshes, and one-dimensional rectilinear, cylindrical and spherical meshes are available. It uses second-order accurate numerical methods to reduce dispersion and dissipation and produce accurate, efficient results [3]. CTH is used extensively at Sandia on all of its high-performance platforms. The results presented are of a scaled study implementation of an actual production “shape charge” problem. The time reported and graphed is referred to as the “grind time” and it is reported in units of seconds per zone-cycle. As the size of the job doubles, the seconds per zone-cycle would half on and ideal machine.



Figures 1a and 1b: CTH Grind Time and Scaling Efficiency

For CTH, optimization level 3 of the Pathscale compiler was failing when compiling CTH. Hence, "-O2 –OPT:Ofast" was used. For this problem set, the PGI compiler exhibited lower execution times at all scale sizes. The speedup was relatively constant, indicating that f_c and f_o for CTH as the problem scales is constant and it is capable of taking advantage of an increase in computational efficiency as the problem scales to larger node counts. This also implies that the parallel inefficiency trend as the problem scales in an algorithmic issue and contributes to an increase in execution time for both the computational and overhead portions of the calculation.

5.2 LAMMPS

The application LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) is a classical molecular dynamics code designed for simulating molecular and atomic systems on parallel computers using spatial-decomposition techniques. [4] LAMMPS is used extensively by the Sandia materials science and molecular science research communities and consumes a significant share of the available node hours on SNL's institutional computing platforms. The problem set used for this study is an example of a biomembrane model developed by Terry Stouch. [4]

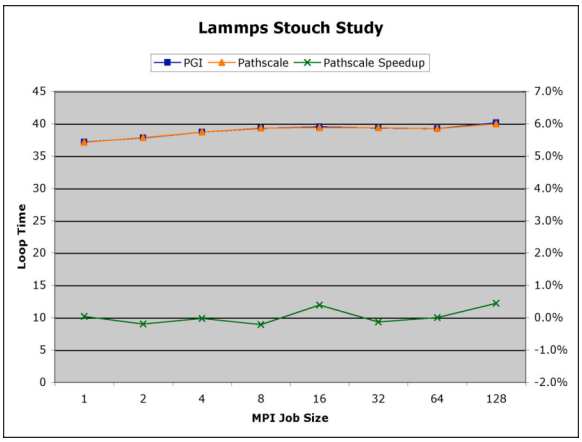


Figure 2: LAMMPS Results

For this problem, the PGI and Pathscale compilers provide essentially equal execution times. Hence, it is difficult to speculate the applications ability to take advantage of node level optimizations as a function of scale.

5.3 PARTISN

PARTISN (Parallel, Time-Dependent SN) is a Los Alamos National Laboratory (LANL) developed application and provides neutron transport solutions on orthogonal meshes with adaptive mesh refinement (AMR) in one, two, and three dimensions. A multigroup energy treatment is used in conjunction with the Sn angular approximation. [5] PARTISN was chosen as a benchmark problem because it exhibits some unique architectural features that stress CPU and message passing resources in a way not exhibited by the other test codes. In particular, it performs a large

number of irregular memory transfers that make its efficiency on a given platform dependent on the memory subsystems performance characteristics. Of course, the efficiency of the memory subsystem is dependent on the instructions used to access memory. A scaled study problem set, SNT48, was used for this analysis. Three timing results are reported: the grind time for a transport calculation, the grind time for a diffusion calculation, and the solver iteration time.

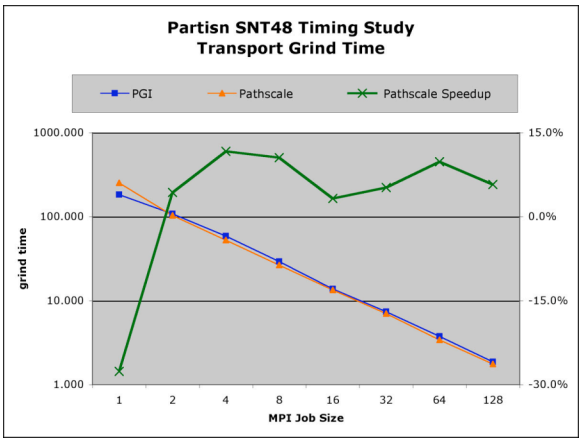


Figure 3: PARTISN Transport Grind Time Results

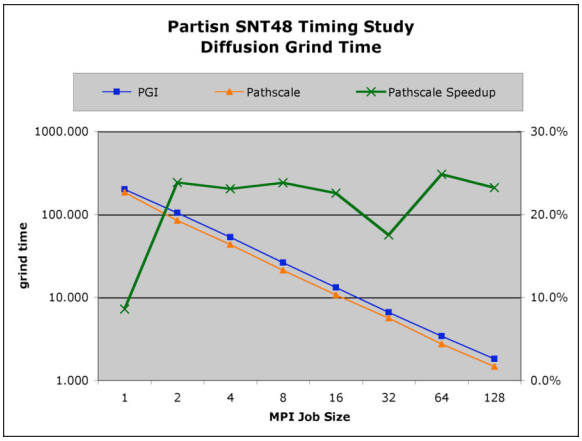


Figure 4: PARTISN Diffusion Grind Time Results

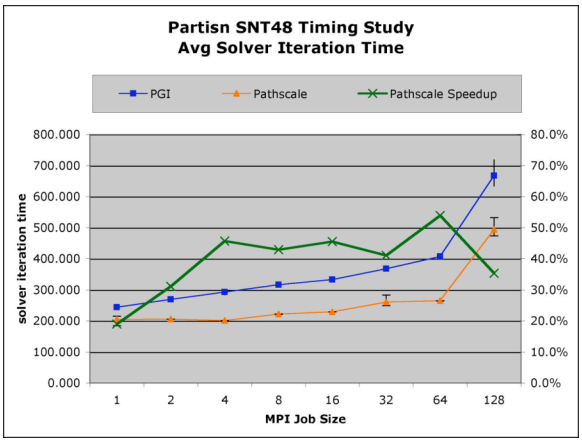
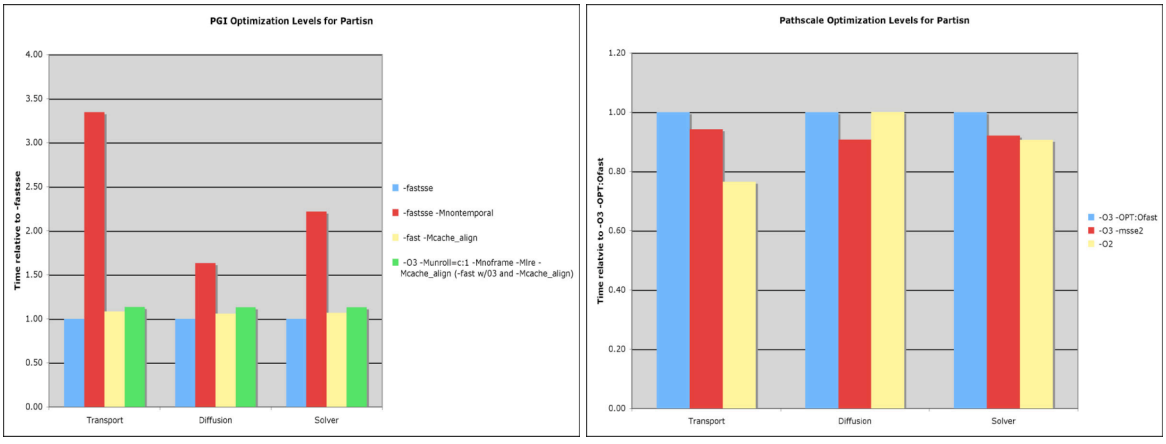


Figure 5: PARTISN Solver Iteration Time Results

Several observations can be made for each timing result. For a single node, the transport grind time for the Pathscale compiler is 28% slower than the PGI compiler calculation. However, as the problem scales to 2 nodes, the Pathscale compiler result becomes 4% more efficient. The Pathscale result remains more efficient, and varies from 3% to 12%, as the problem scales to 128 nodes. Ideally, as the size of the problem doubles the grind time is halved. Inefficiencies and algorithmic issues may keep the slope of the grind time plot to be less than half, but the trend should still be linear. This is observed in the PGI compiler result from 2 to 128 nodes, but for some reason the single node result is much faster than that of the Pathscale result. A similar trend is noticed in the diffusion calculation, but not to the same degree. At one node, the Pathscale result is 9% faster, but at 2 nodes it jumps to 24% and remains in that same neighborhood up to 128 nodes, with the exception of the 32-node problem size where the speedup falls to 18%. The Pathscale compiler provides a speedup of 19% to 54% for the solver iteration time. As the problem scales, both results show a linear increase in iteration time until the 128-node result is reached. It is believed that the nonlinear increase in iteration time at the jump from 64 to 128 nodes is an artifact of Red Squall's Quadrics network architecture. At a job size of more than 64 nodes, messaging is performed beyond a single, fully connected, 64-port switch chassis and higher level network layer, with one-half of the full connectivity in the fat tree network, is used to access the next 64-port switch.

In order to better understand the differences between the PGI and Pathscale results, different optimization levels were tried for each compiler. Since PARTISN uses a lot of irregular memory references it was believed that perhaps the SSE type instructions, which are for the most part vector calculation optimizations, enabled by the "-fastsse" switch may be an issue. The first test is a proof by counter example in which an additional optimization switch was added, "-Mnontemporal". This option allows the compiler to use nontemporal prefetching instructions, and in particular the Opteron's prefectnta instruction. This instruction reads a given number of bytes into L1 cache, and that data is never evicted to the L2 cache and hence avoids cache pollution. This technique works well for large, unit stride, data accesses where the data will not be reused. This is the opposite of what is required for irregular access patterns such as those seen in PARTISN, so by enabling it is expected that the run times will increase. When executed on a single node, the transport grind time increased by more than a factor of 3, diffusion grind time increased by a factor of ~1.5 and the iteration solver time increased by a factor of greater than 2. This shows that PARTISN does not profit from nontemporal operations and validates the assertion that it is heavily based on irregular memory references. The second test was to remove SSE related optimizations from "-fastsse". This was accomplished by only keeping the "-fast" and "-Mcache_align" flags and removing SSE related flags. This results in a slight increase in run time for the three calculations. The "-fast" switch uses an optimization level of "-O2". The final test was to use "-O3" and the other flags found in "-fast". These settings also show slightly slower runtimes for all calculations. It was concluded that "-fastsse" alone provides the most optimal result for the PGI compiler.

Similar sets of tests were performed for the Pathscale compiler. The first was to remove the "-OPT:Ofast" switch but keep the "-msse2" flag. This reduced the runtime for the transport calculation by a factor of ~0.9. Reducing the optimization to just the use of the "-O2" flag reduced the single node performance even further for the transport and solver times, a factor of 0.77 and 0.91 respectively. The diffusion calculation time remained the same. Scaled tests were performed with the Pathscale version using "-O2", and the results are shown in Figures 7, 8 and 9.



Figures 6a and 6b: Different Optimization Level Results for a Single Node

Single node performance for the Pathscale results were improved and the trend in the graphed results now behave similar to the PGI results. However, the improvements in runtime at scale are not the same magnitude as with "-O3 –OPT:Ofast" and in some cases the PGI compiler is faster. The transport grind time results show the PGI compiler

providing better or equal performance for all job sizes, with the exception of the 2-node result. For the diffusion calculation, the Pathscale result is faster, except for the 8 and 32 node results. Solver iteration time is less than that of the PGI result for all job sizes. All three calculations exhibit a behavior in which the 8, 32 and, with the exception of the transport calculation, 128 node result less efficient for the Pathscale compiler than for the other job sizes. Although this trend is observed in the results using "-O3 –OPT:Ofast", results for the "-O2" tests are more pronounced. It is assumed that the method for how the initial data set is partitioned across the nodes uncovers inefficiency with the Pathscale compiled code.

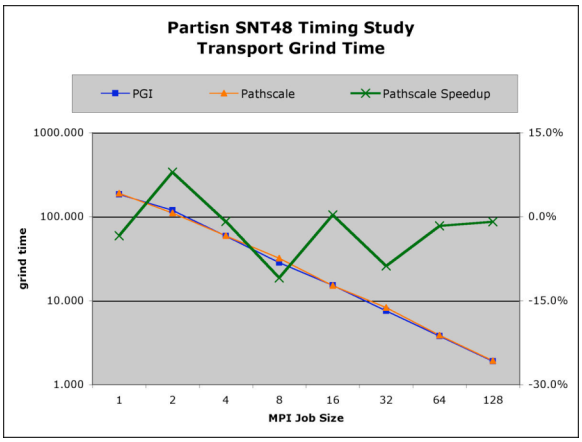


Figure 7: PARTISN Transport Grind Time Results w/Pathscale –O2

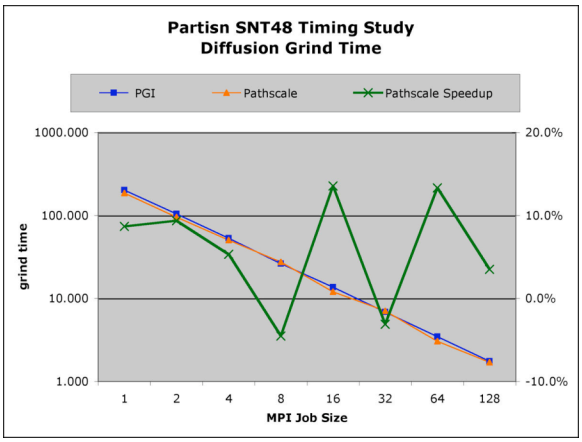


Figure 8: PARTISN Diffusion Grind Time Results w/Pathscale –O2

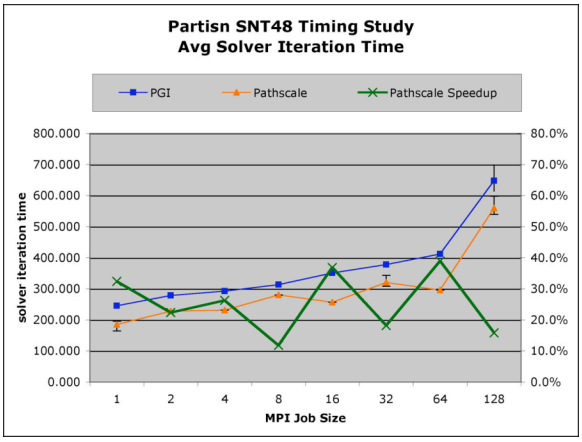


Figure 9: PARTISN Solver Iteration Time Results w/Pathscale –O2

In general, the PARTISN code seems to be able to take advantage of code optimizations as the scale of the problem grows. However, there seems to be algorithmic and network issues involved as the problem scales and hence the magnitude of the optimization advantage is dependent on the size of the job.

6 Conclusions

The PGI compiler showed better execution times for CTH. The Pathscale compiler provided a significant performance advantage for PARTISN. The LAMMPS study provided no advantage for either compiler. It was shown that CTH takes advantage of increased compiler optimizations at scale and the results demonstrated that as CTH scales the fraction of computational work and the fraction of overhead are relatively constant. Hence, the decreasing parallel efficiency must be due to algorithmic effects. The PARTISN results show that the magnitude of the performance advantage for Pathscale is dependent on job size, but even at a scale of 128 nodes when network issues were involved, the advantage was significant.

Compiler deployment is a significant part of any HPC system and serious considerations should be taken when deciding on what compiler to choose for your platform. It has been shown that multiple compiler choices would be beneficial to the application developer. The cost of a compiler and it's licenses for an HPC platform can be a significant investment, but an increase in efficiency for a given code over a given number of nodes hours may make it cost beneficial.

References

[1] http://www.pgroup.com, The Portland Group Web Site
[2] http://www.pathscale.com, Pathscale Web Site
[3] http://www.cs.sandia.gov/web9232/cth, CTH Web Site
[4] http://www.cs.sandia.gov/~sjplimp/lammps.html, LAMMPS Web Site
[5] http://www.ccs.lanl.gov/CCS/CCS-4/code.shtml, PARTISN Web Site